Apellidos:

Nombre:

Dni:

Examen de Procesadores de Lenguaje

Septiembre de 2007
(2 horas y 45 minutos)

Instrucciones

- Entrega esta hoja, rellenada con tus datos, junto a tu examen cuando lo entregues.
- Ten en cuenta que es requisito para aprobar haber superado previamente las prácticas de la asignatura de esta convocatoria.
- Pon en el encabezado de todas las hojas que entregues tus apellidos y tu nombre.
- Si estás en 5.ª o 6.ª convocatoria, pon una cruz a continuación: ☐
- Recuerda que puedes escribir tu respuesta tanto en valenciano como en castellano.
- Has de entregar tus respuestas por separado como sigue: por un lado, la solución a la cuestión C1 y al problema P1; por otro lado, la solución a los problemas P2 y P3. Dentro de cada grupo no es necesario entregar la respuesta a cada pregunta en una hoja separada.
- La respuesta a la pregunta C1 ha de ser correcta para poder superar el examen: di exactamente qué has hecho en tu práctica, sin comentarios ni explicaciones adicionales.

Preguntas

C1 En la especificación del lenguaje fuente del compilador de tu práctica se afirma que no puede usarse la instrucción `return` dentro del cuerpo del programa principal. ¿A qué se refiere con esto? Pon un ejemplo. ¿Cómo has detectado en el código de tu compilador esta circunstancia para emitir el error oportuno?

P1 (5 PUNTOS)

Considera el siguiente fragmento de la especificación sintáctica de un lenguaje:

$$
\begin{array}{lcl}
I & \longrightarrow & \textbf{print}\ E \\
E & \longrightarrow & E\ \textbf{or}\ T \\
E & \longrightarrow & T \\
T & \longrightarrow & T\ \textbf{and}\ F \\
T & \longrightarrow & F \\
F & \longrightarrow & \textbf{id} \\
F & \longrightarrow & \textbf{cteint} \\
F & \longrightarrow & \textbf{id opinc} \\
F & \longrightarrow & \textbf{opinc id} \\
F & \longrightarrow & (\ E\ )
\end{array}
$$

Diseña un esquema de traducción dirigida por la sintaxis (ETDS) que genere el código CIL adecuado para la máquina virtual de la plataforma .NET y que emita los mensajes de error semántico oportunos. Para ello, has de tener en cuenta los siguientes aspectos:

a. El único tipo de dato simple de este lenguaje es el tipo entero.

b. El único lexema asociado al *token* **opinc** es `++`.

c. Esta parte del lenguaje se encarga principalmente de subexpresiones lógicas en las que las variables pueden venir acompañadas de operadores de preincremento y postincremento.

d. Cuando el operador de incremento precede al operando, el resultado de la subexpresión será el valor obtenido tras realizar la operación; en otro caso, será el valor del operando justo antes de realizarla. El valor resultante de la operación de incremento deberá almacenarse en ambos casos en la variable que actúa como operando.

   El comportamiento es análogo al que se produce en Java con estos operadores: por ejemplo, si consideremos el operador de suma y que la variable entera `a` vale 1, el resultado de la expresión `a++ + a++` ha de ser 3 $(1 + 2)$; por tanto, el código del incremento se ha de ejecutar antes de la suma, en este caso. Otro ejemplo: si `a` vale 1, el resultado de la expresión `++a + ++a - a++` es $2+3-3 = 2$, y al acabar la evaluación de la expresión la variable `a` valdrá 4.

e. De cara a los operadores lógicos, un valor entero de `0` equivale a *falso* y cualquier otro valor a *cierto*. Estos operadores devuelven un `0` o un `1` según el resultado sea *falso* o *cierto*.

f. Los operadores lógicos se han de evaluar en *cortocircuito*, es decir, el segundo operando solo se evalúa (en tiempo de ejecución) si el primero no es suficiente para determinar el valor de la expresión: cuando el primer

argumento de `a and b` es *falso*, el valor final es *falso*; cuando el primer argumento de `a or b` es *cierto*, el valor final es *cierto*. Esto tiene consecuencias importantes en expresiones como, por ejemplo, `a or b++ or ++c`, porque el incremento de las variables no llega a realizarse cuando la variable `a` vale *cierto*.

g. El ETDS ha de realizar los cálculos que determinen el tamaño más ajustado de la pila (necesarios para la directiva `maxstack`). Debido a la evaluación en cortocircuito de los operadores booleanos, es posible que el tamaño máximo que obtengas en tiempo de compilación sea superior al necesario para una ejecución particular.

h. Para simplificar, considera que el lenguaje fuente exige que todas las instrucciones aparezcan dentro de los métodos (no hay por lo tanto un programa principal independiente de cualquier clase).

i. Asume que la tabla de símbolos actual es accesible a través de la referencia global `TSA`.

j. Para simplificar, considera que todas las variables son locales (no has de considerar, por tanto, ni parámetros ni atributos).

k. *Nota:* dado que las acciones semánticas de las reglas de E y de T serán muy parecidas, no es necesario que las escribas todas; es suficiente con que escribas con detalle la parte del ETDS de las reglas de E y después comentes adecuadamente los elementos que serían diferentes en las reglas de T. Estos comentarios han de dejar claro a un posible implementador del ETDS que no entienda el problema los pasos que ha de seguir para obtener las acciones semáticas de T a partir de las de E.

l. *Nota:* las instrucciones `and` y `or` del repertorio de instrucciones de CIL funcionan a nivel de bit y, por lo tanto, no siguen la semántica de los operadores lógicos de este lenguaje.

m. La signatura del método del API de .NET que imprime un entero es `void [mscorlib]System.Console::Write(int32)`.

Señala claramente qué hace cualquier función auxiliar que utilices. Toda la información debe pasarse a través de atributos y no es posible usar ninguna variable global (excepto las correspondientes a la tabla de símbolos y de tipos), pero sí variables locales a una regla. Indica, además, de qué tipo es cada uno de los atributos que utilizas y cuál es su cometido.

Nota: recuerda utilizar la notación de los ETDS y *no* la de `yacc`.

P2 (2,5 PUNTOS)

Una subgramática que describe la estructura de la instrucción condicional empleada en muchos lenguajes de programación es la siguiente (la regla del `print` se añade por dar completitud a la gramática):

$$
\begin{array}{lcl}
\mathit{Instr} & \longrightarrow & \textbf{if} \; (\; E \;) \; \mathit{Instr} \\
\mathit{Instr} & \longrightarrow & \textbf{if} \; (\; E \;) \; \mathit{Instr} \; \textbf{else} \; \mathit{Instr} \\
\mathit{Instr} & \longrightarrow & \textbf{print}
\end{array}
$$

a) La gramática anterior es ambigua. ¿Por qué? Añade algún ejemplo a tu justificación.

b) Aunque existen formas no ambiguas parcialmente equivalentes, la gramática anterior suele ser la que se implemente en muchos analizadores sintácticos, ya que es más sencilla e intuitiva. Centrándonos en el caso de los analizadores sintácticos ascendentes SLR, esto genera un conflicto en las tablas resultantes. ¿Por qué? ¿Cómo se *esquiva* este conflicto? Añade un ejemplo a tu justificación.

c) Finalmente, otros lenguajes utilizan unas marcas especiales para identificar el final de cada instrucción condicional con lo que se elimina el problema de la ambigüedad. ¿Cómo lo hacen? Da un ejemplo de las reglas de la gramática en este caso.

P4 (2,5 PUNTOS)

Indica el código que se generaría para el lenguaje objeto `m2r` ante el siguiente fragmento de programa fuente. Considera que el método `int g (int)` pertenece a la misma clase que el método `f`. Comenta brevemente la acción que realiza cada secuencia de instrucciones objeto que escribas.

```
...
public method int f (int m, int n) {
  return 1+g(m+n)*2;
}
...
```